A spring search algorithm applied to engineering optimization problems

Dehghani, Mohammad; Montazeri, Zeinab; Dhiman, Gaurav; Malik, O. P.; Morales-Menendez, Ruben; Ramirez-Mendoza, Ricardo A.; Dehghani, Ali; Guerrero, Josep M.; Parra-Arroyo, Lizeth

Published in:
Applied Sciences (Switzerland)

DOI (link to publication from Publisher):
10.3390/APP10186173

Publication date:
2020

Document Version
Publisher's PDF, also known as Version of record

applied sciences

MDPI

Article

A Spring Search Algorithm Applied to Engineering Optimization Problems

Mohammad Dehghani [1], Zeinab Montazeri [1], Gaurav Dhiman [2], O. P. Malik [3], Ruben Morales-Menendez [4], Ricardo A. Ramirez-Mendoza [4,*], Ali Dehghani [5], Josep M. Guerrero [6] and Lizeth Parra-Arroyo [4]

1 Department of Electrical and Electronics Engineering, Shiraz University of Technology, Shiraz 71557-13876, Iran; m.dehghani@sutech.ac.ir (M.D.); Z.Montazeri@sutech.ac.ir (Z.M.)
2 Department of Computer Science, Government Bikram College of Commerce, Patiala, Punjab 147004, India; gaurav.dhiman@thapar.edu
3 Department of Electrical and Computer Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada; maliko@ucalgary.ca
4 School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey 64849, Mexico; rmm@tec.mx (R.M.-M.); A01036078@itesm.mx (L.P.-A.)
5 Department of Civil Engineering, Islamic Azad Universities of Estahban, Estahban 74, Iran; adanbax2@gmail.com
6 CROM Center for Research on Microgrids, Department of Energy Technology, Aalborg University, 9220 Aalborg, Denmark; joz@et.aau.dk
* Correspondence: ricardo.ramirez@tec.mx; Tel.: +52-81-2001-5597

Received: 18 August 2020; Accepted: 2 September 2020; Published: 4 September 2020

Abstract: At present, optimization algorithms are used extensively. One particular type of such algorithms includes random-based heuristic population optimization algorithms, which may be created by modeling scientific phenomena, like, for example, physical processes. The present article proposes a novel optimization algorithm based on Hooke's law, called the spring search algorithm (SSA), which aims to solve single-objective constrained optimization problems. In the SSA, search agents are weights joined through springs, which, as Hooke's law states, possess a force that corresponds to its length. The mathematics behind the algorithm are presented in the text. In order to test its functionality, it is executed on 38 established benchmark test functions and weighed against eight other optimization algorithms: a genetic algorithm (GA), a gravitational search algorithm (GSA), a grasshopper optimization algorithm (GOA), particle swarm optimization (PSO), teaching–learning-based optimization (TLBO), a grey wolf optimizer (GWO), a spotted hyena optimizer (SHO), as well as an emperor penguin optimizer (EPO). To test the SSA's usability, it is employed on five engineering optimization problems. The SSA delivered better fitting results than the other algorithms in unimodal objective function, multimodal objective functions, CEC 2015, in addition to the optimization problems in engineering.

Keywords: heuristic algorithms; optimization; spring force; spring search; spring

1. Introduction

As the demand for quick and accurate solutions to ever increasingly complex problems expands, classical methods are being substituted for more robust approaches. One proposal is the use of heuristic random-based algorithms in place of searching the defined problem space exhaustively [1–5]. Heuristic algorithms are applicable to a variety of scientific fields, such as: logistics [6], bioinformatics [7], data mining [8], chemical physics [9], energy [10], security [11], electrical engineering [12–16], energy carriers [17,18] as well as other fields that aim to discover the optimal solution.

Each population-based algorithm may represent the conveying of data along with the interlinkage between elements in a different way. For example, genetic algorithms simulate evolution [19] while annealing algorithms, thermodynamics [20]; immunity algorithms, the human immune system [21]; colony optimization strategies, ants' search for food [22]; and particle swarm optimization approaches, the behavior of birds while searching for food [23].

There are many laws of nature that may serve as inspiration, such as Newton's universal law of gravitation, Hooke's spring law, the laws of motion, the laws of energy and mass conservation, as well as the laws that dictate the electromagnetic force. A novel optimization algorithm was proposed based on Hooke's spring law, with its corresponding precursory results detailed in [24]. Such algorithm is detailed and analyzed in the current paper with its improved equations. The *SSAs* capabilities were evaluated through 23 benchmark test functions, as well as a group of problems mentioned in the Constrained Single Objective Real-Parameter Optimization Technical Report, 'CEC'2015'. The SSA was further corroborated by being weighed against eight established algorithms found in the literature, as well as being used to solve a selection of engineering problems.

Section 2 provides a greater insight to other established optimization approaches. Section 3 elucidates Hooke's law while Section 4 outlines the SSA. Section 5 assesses the algorithm's search ability while Section 6, its proficiency. Section 7 includes the outcome of the evaluation through the standard benchmark test functions. Section 8 includes the implementation of the algorithm on select engineering design problems. Finally, Section 9 encompasses conclusions.

2. A Brief History of Intelligent Algorithms

An algorithm is considered intelligent when it finds a suitable answer or solution to a problem whose main characteristic is optimization in the shortest possible time and using the least amount of available information [25]. Using a more complete definition, the heuristic method is a strategy that sacrifices part of the information to reach a solution in the shortest possible time and with good precision [26]. Usually, heuristic algorithms are very frequently based on natural processes, that is, typically biological processes or laws that explain physical phenomena. This approach has been widely considered in the last ten years, and numerous algorithms have been suggested. These algorithms have been classified into different categories, such as swarm-based algorithms, evolution-based algorithms, and physics-based algorithms.

2.1. Swarm-Based Algorithms

These techniques were developed from the analysis of several processes that exist naturally, such as the growth or symbiosis of plants, the feeding behavior of insects, and the behavior and social organization of animals [27]. The particle swarm optimization (PSO) algorithm is an indeterminate (random) search method that was developed around 1995 to support functional optimization [28]. This algorithm was developed by analyzing and taking a reference to the movement that birds develop as a group (team) when looking for food. The algorithm is based on the premise that a group of birds looks for food at random and that there is only one portion of food in the area (search space) in question, but none of the birds know where the food is. One of the most successful strategies could be: for the birds to follow the bird that is closest to the food, and in sequence to be the bird most likely to find the food. This strategy is, in fact, the source of the algorithm. In the algorithm, each solution, called a particle, is equivalent to a bird in the bird movement algorithm. Each particle (bird) has an arbitrary value calculated by a success function. Each particle (bird) also has a speed that controls the particle (bird). By continuing to search for optimal particles, the agent continues to move in the solution space. The firefly algorithm (FA) is an algorithm inspired by a natural system; in this case, based on swarms for projects where limited optimal solutions are sought [29]. The algorithm is inspired by the analysis of the radiation behavior of these insects. The firefly lives in groups and changes from low light to higher light intensity. The firefly algorithm generates a rhythmic light and passes through each different light pattern or behavior among these insects. The firefly algorithm uses

these lights for two main purposes: finding mates to mate with and looking for food. These lights can also serve as a protection mechanism or strategy. The whale optimization algorithm (WOA) [30] is another nature-inspired optimization algorithm; as the name implies, this algorithm mimics the social behavior that humpback whales have. The most surprising thing about humpback whales is their hunting strategy. This food search strategy is called a bubble net feeding method. Humpback whales like to hunt schools of krill or small fish near the surface of the ocean. It has been analyzed that this foraging is done by creating distinctive bubbles along a circle or route in the form of the number "9". Some of the other swarm-based algorithms are: artificial bee colony (ABC) [31], bat-inspired algorithm (BA) [32], spotted hyena optimizer (SHO) [33], cuckoo search (CS) [34], monkey search (MS) [35], group optimization (GO) [36], artificial fish-swarm algorithm (AFSA) [37], hunting search (HS) [38], moth-flame optimization algorithm (MFO) [39], dolphin partner optimization (DPO) [40], orientation search algorithm (OSA) [41], binary orientation search algorithm (BOSA) [42], dice game optimizer (DGO) [43], shell game optimization (SGO) [44], hide objects game optimization (HOGO) [45], donkey theorem optimization (DTO) [46], following optimization algorithm (FOA) [47], rat swarm optimizer (RSO) [48], darts game optimizer (DGO) [49], football game based optimization (FGBO) [50], grey wolf optimizer (GWO) [51], grasshopper optimization algorithm (GOA) [52], coupled spring forced bat algorithm (SFBA) [53], adaptive granularity learning distributed particle swarm optimization (AGLDPSO) [54], multi leader optimizer (MLO) [55], doctor and patient optimization (DPO) [56], and emperor penguin optimizer (EPO) [57].

2.2. Evolution-Based Algorithms

An algorithm is considered evolutionary when the algorithm combines aspects of natural selection and continuity of coordination. These algorithms are based on structures that simulate the rules of selection, recombination, change, and survival, similar to genetics, hence the adjective algorithm. These structures are based on genetic sets. In this method, the environment determines each person's coordination or performance in a population and uses the most consistent individuals to reproduce [58]. Evolutionary algorithms are random search procedures that use genetic mechanisms and natural selection [59]. Genetic algorithms (GA) were developed as a method that seeks optimization, starting from fundamental basic operations in genetic biology [60]. The first record of using these concepts to create an optimization method occurred in 1967 [61]. GA is a particular type of evolution algorithm that exploits basic biological concepts such as inheritance and mutation [62] and has had satisfactory results in different scientific domains.

On the other hand, differential evolution (DE) [63] is an algorithm that also seeks intelligent optimization based on populations introduced in 1995 [64]. The initial version of this algorithm was used to solve problems with continuous variables, and interpretations of this algorithm have been implemented over time to solve optimization problems with discrete variables [65]. Other algorithms based on the theory of evolution have been developed, examples of which are: evolutionary programming (EP) [66], biogeography-based optimizer (BBO) [67], enhanced quantum-inspired differential evolution algorithm (IQDE) [68], genetic programming (GP) [69] and evolution strategy (ES) [70].

2.3. Physics-Based Algorithms

Physics-based algorithms, as the name implies, are inspired by the laws of physics. Simulated annealing (SA) is one of the best known and most popular optimization algorithms. SA was developed in 1983 [71], inspired by metals' annealing, for example, in the artisan process to make swords or knives in the past. The process consists of first heating the metal to a very high temperature and then cooling it by gradually reducing the temperature so that the metal hardens and becomes harder. In this process, when the temperature of the metal increases, the speed of atomic movement increases dramatically and, in the next step, the gradual reduction in temperature causes the formation of specific patterns based on the location of its atoms [20]. Drastic temperature change is one of the adjustment parameters

of this algorithm. The gravitational search algorithm (GSA) [72] is inspired by the universal law of gravitation developed by Isaac Newton. In this algorithm, objects such as planets in a galaxy are defined as search agents. The optimal region, similar to a black hole, absorbs the planets. Information about the fitness of any object is stored as gravity and mass inertia. The exchange of information and the effects of objects on each other is governed by the attractive force of gravitation [73]. Several algorithms have been developed based on laws and/or theories of physics, such as: charged system search (CSS) [74], galaxy-based search algorithm (GBSA) [75], curved space optimization (CSO) [76], ray optimization (RO) algorithm [77], artificial chemical reaction optimization algorithm (ACROA) [78], small world optimization algorithm (SWOA) [79], central force optimization (CFO) [80], black hole (BH) [81] and big-bang big-crunch (BBBC) [82].

3. Spring Force Law

If a force that moves an object in a closed path (forward and backward) is not affected by the object's trajectory, that force is conservative. Another method for diagnosing conservative forces is that the work done by the force in different paths is equal to the difference between the initial and final points. The spring force is a type of conservative force [83].

Consider a spring that imposes a force on a particle with mass 'm'. The particle moves horizontally in the x direction. When a particle is at the origin ($x = 0$), the spring is balanced. An external force (F_{ext}) influences the object in the anti-clock wise direction of the spring. The external force is always equal to the spring force. Thus, the particle is always in balance.

Consider that the particle is moved a distance x from its initial location $x = 0$. When the external factor imposes a force F_{ext} on the particle, the spring offers a resistance force F_s on the particle. This force can be described by the spring force or Hooke's law as follows:

$$F_s = -kx \tag{1}$$

where k is the spring force constant and x denotes the spring displacement (strain or compression) from the balance point. Most real springs properly follow Hooke's law up to a limit [84].

The system behavior is investigated in an isolated system as shown in Figure 1. It is assumed that only the spring force is imposed on the object.

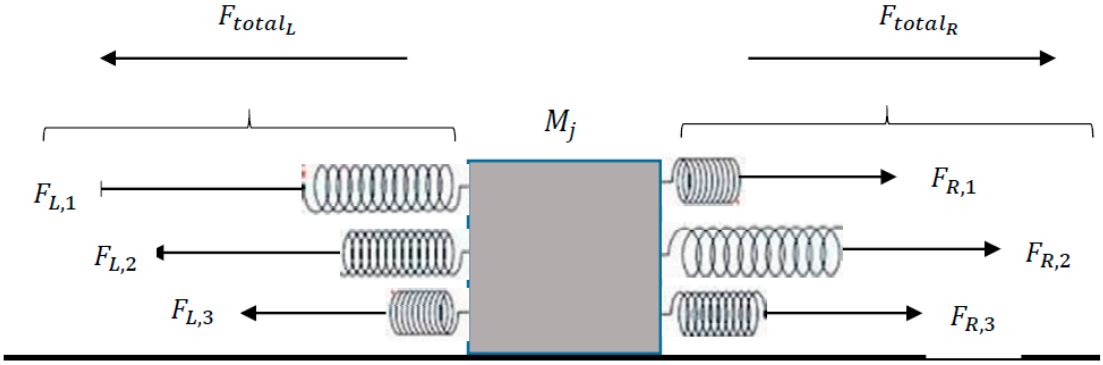

Figure 1. An isolated system is composed of object and spring forces.

In Figure 1, forces imposed on object j can be grouped into two, F_{total_R}, which variables as described in Equation (2), is the sum of forces imposed from the right and F_{total_L}, which, as shown in Equation (3), is the sum of forces imposed from the left. It is necessary to mention that the springs that are attached to the object from either right or left, are also attached to robust points at their other ends.

$$F^j_{total_R} = \sum_{i=1}^{n_R} K_{i,j} x_{i,j} \tag{2}$$

$$F^j_{total_L} = \sum_{l=1}^{n_L} K_{l,j} x_{l,j} \tag{3}$$

where, n_R and n_L are the number of left and right spring forces, $x_{i,j}$ and $x_{l,j}$ show the distance between the object j and the fixed left and right points, $K_{i,j}$ and $K_{l,j}$ are the spring stiffness coefficients between the object j and the fixed points.

The object is initially balanced with no force exerted on it. Then, by applying the spring forces, the object is pulled from the right and the left. Considering the magnitude of these forces, the spring either shifts to left or right until the system reaches a new equilibrium position. It is apt to mention that if the right and left forces are equal, the object remains at its original position.

Considering the stiffness coefficient of the springs that are connected to the object, two new parameters may be defined as below:

$$K^j_{equal_R} = \sum_{i=1}^{n_R} K_{i,j} \tag{4}$$

$$K^j_{equal_L} = \sum_{l=1}^{n_L} K_{l,j} \tag{5}$$

$K^j_{equal_R}$ and $K^j_{equal_L}$ are the right and the left constants of the spring, respectively. Considering Equation (1) the displacement values at each side may be defined as follows:

$$dX^j_R = \frac{F^j_{total_R}}{K^j_{equal_R}} \tag{6}$$

$$dX^j_L = \frac{F^j_{total_L}}{K^j_{equal_L}} \tag{7}$$

here, dX^j_R and dX^j_L are the displacement values of object j to the right and left, respectively. Therefore, the total displacement may be defined as follows:

$$X^j = dX^j_R + dX^j_L \tag{8}$$

dX^j is the final object j displacement value that may be a positive or negative value.

$$X^j = X^j_0 + dX^j \tag{9}$$

Equation (9), X^j relates to the location of the new balance point of the system and object j. Besides, X^j_0 is the initial balance of object j.

By simulating Hooke's law within the discrete time domain, a new optimization algorithm called the spring optimization was designed, which is explained further in the following section.

4. Spring Search Algorithm (SSA)

In this article, the spring search algorithm is run in an artificial system with discrete time. The system space is the defined domain of problem. It is possible to apply the spring force law as a tool to convey information. The designed optimization may be applied to solve any optimization problem, as long as the answer can be defined as a position within space, and its similarity to other problem answers can be expressed as spring stiffness comparisons. The stiffness of the spring is established relative to the objective function.

SSAs consist of two general steps: making a discrete time artificial system within the problem space by defining the initial positioning of objects, determining the governing laws and arranging parameters; letting the algorithm run until it reaches a stop.

4.1. Setting the System, Determining Laws and Arranging Parameters

First, the system space is determined with its multi-dimensional coordinate system in which the problem is defined. Any point in space may be the solution of the optimization problem. Search factors are sets of objects that are attached to each other by some springs. Each object has its own position as well as stiffness coefficients pertaining to the springs attached to it. The object's position is a point in space where the solution of the problem may be found. The values of the springs are computed regarding the stiffness of both objects attached.

After setting the system, its governing laws are determined. Only the spring force laws and movement laws are observed in the system. The general patterns of these rules are similar to natural laws and are defined below.

In physics, mechanical and elastic science, Hooke's law is an approximation that shows that any change in an object's length is directly proportional to its load. Most materials follow this law with reasonable accuracy, except when the force is lower than its elasticity. Any deviation from Hooke's law increases with the deformation quantity, such that with numerous deformations, when the material trespasses its linear elastic domain, Hooke's law loses its application. The present article assumes that Hooke's law is valid for all of the time observed.

The present location of any object equals the sum coefficient of its previous locations and its displacements according to the laws of motion. Any object displacement may be determined with the aid of the spring force law.

Consider a system as a set of m objects where the position of each object is a point in the search space and a solution to the problem.

$$X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^n\right) \tag{10}$$

The position of an object i of dimension d is designated x_i^d in Equation (10). The initial positions of the objects are defined within the search space randomly. These objects tend to return to an equilibrium position by means of the forces exerted by the spring.

Equation (11) is employed in order to compute the spring stiffness coefficient.

$$K_{i,j} = K_{max} \times \left|F_n^i - F_n^j\right| \times \max\left(F_n^i, F_n^j\right) \tag{11}$$

In Equation (11), $K_{i,j}$ is the spring stiffness coefficient among objects i and j, K_{max} is the maximum value of the spring stiffness coefficient, and is determined according to the type of problem in question, F_n^i and F_n^j are the normalized objective functions of objects i and j respectively. Equations (12) and (13) are used in order to normalize the objective function.

$$F'^i_n = \frac{f_{obj}^i}{\min\left(f_{obj}\right)} \tag{12}$$

$$F_n^i = \min\left(F'^i_n\right) \times \frac{1}{F'^i_n} \tag{13}$$

In the above equations, F_{obj} is the objective function and f_{obj}^i is the objective function value for the object i.

In the m variable problem, it is possible to assume that the problem has m dimensions and that there is a coordinate for each dimension. Therefore, it is possible to draw a system on its related coordinate based on each variable. Each coordinate's strong points at the left or right side of the object

are determined by comparing the objective function quantities. Stronger points related to each object mean that they are positioned at more optimal positions. Therefore, each coordinate has two general summative forces: the sum of the right, Equation (14) and the sum of the left, Equation (15). Both are applied to the object j.

$$F^{j,d}_{total_R} = \sum_{i=1}^{n^d_R} K_{i,j} x^d_{i,j} \tag{14}$$

$$F^{j,d}_{total_L} = \sum_{l=1}^{n^d_L} K_{l,j} x^d_{l,j} \tag{15}$$

In the above equations, $F^{j,d}_{total_R}$ stands for the sum of the right forces and $F^{j,d}_{total_L}$ the sum of the left forces imposed on object j within the dimension d. n^d_R and n^d_L are the d dimension right and left strong points; $x^d_{i,j}$ and $x^d_{l,j}$ represent the distance of the object 'j' from the right and left strong points; $K_{i,j}$ and $K_{l,j}$ are the spring stiffness coefficient attached to the object j and the strong points.

Now, considering Hooke's law in the dimension d:

$$dX^{j,d}_R = \frac{F^{j,d}_{total_R}}{K^j_{equal_R}} \tag{16}$$

$$dX^{j,d}_L = \frac{F^{j,d}_{total_L}}{K^j_{equal_L}} \tag{17}$$

here, $dX^{j,d}_R$ and $dX^{j,d}_L$ are the displacement of object j to the right and the left of dimension d, respectively. The total displacement may be calculated as follows:

$$dX^{j,d} = dX^{j,d}_R + dX^{j,d}_L \tag{18}$$

where $dX^{j,d}$ is the final displacement of object j in the dimension d. The direction may be in the positive or negative x direction.

$$X^{j,d}_{new} = X^{j,d}_0 + r_1 \times dX^{j,d}, \tag{19}$$

Equation (19) relates $X^{j,d}_{new}$ to both the new position and balance point of the system, along with the d dimension of object j. Additionally, $X^{j,d}_0$ is the initial balance point of object j along with the d dimension. Here, r_1 is a random number with a uniform distribution within the span of [0–1], which is used to preserve the random mode.

In the last step, the objects and springs after reaching balance have a small displacement due to slipping, which is simulated in Equation (20).

$$X^{j,d} = X^{j,d}_{new} + \frac{2 \times (T - t)}{T} \times (-0.2 + r_2 \times 0.4) \times X^{j,d}_{new} \tag{20}$$

here, $X^{j,d}$ is the updated position of d dimension of object j, T is the maximum number of iteration, t is the counter of iteration, and r_2 is a random number with a uniform distribution within the span of [0–1], which is used to preserve the random mode.

4.2. Time Passing and Parameters Updating

While initially forming a system, each object is randomly placed within a point in space where it may be the problem's answer. At any given time, objects are assessed and their displacement is calculated using Equations (11) through (18). Thereafter, the object is placed in the new computed

position. The parameter of interest is the spring stiffness coefficient that is updated at each stage based on Equation (11). The stop condition is established after the algorithm has been run for a given time. The steps of the SSA are as follows and a flowchart encompassing them is shown in Figure 2:

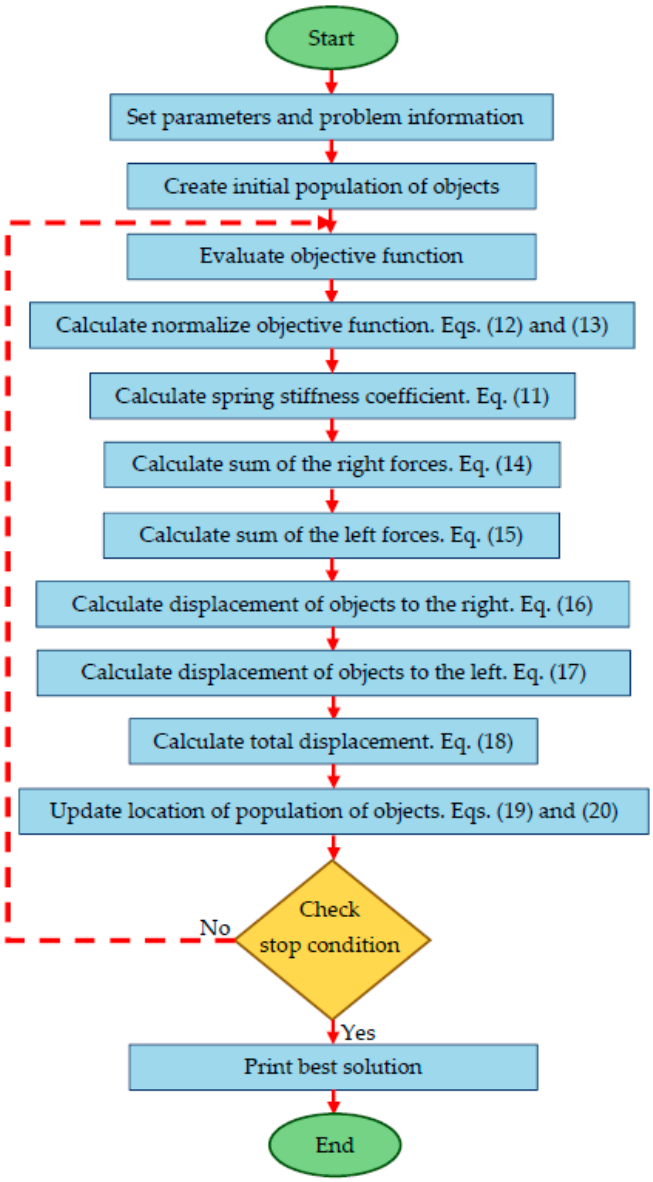

Figure 2. Spring Search Algorithm Flowchart.

1- Start
2- Determine the system environment and the problem information
3- Create the initial population of objects
4- Evaluate and normalize fitness function or objective function
5- Update parameter K
6- Formulate the spring force and the laws of motion for each object
7- Compute object displacement quantities
8- Update object locations
9- Repeat steps 4 through 8 until the stop condition is satisfied
10- Print best solution
11- End

5. Properties of the Proposed *SSA*

The above algorithm is a proposed optimization method that applies the spring force law. The strategy of the algorithm proposed is to use the spring fitness coefficient. In this algorithm, a set of objects search the space randomly using the spring force as a tool to transfer information between objects. Under the influence of other objects, each may arrive at a rough understanding of its surrounding space. The algorithm must be navigated such that the objects' location improves as time passes.

Thus, springs with more fitness coefficients can be attached to those with better fitness functions. The springs attract other objects to themselves, allowing a suitable force to be exerted on each object. Objects tend to move towards better conditions as time goes by. Accordingly, objects placed in better locations must take slower and shorter steps. As an object arrives in a better condition, its stiffness coefficient increases. The stiffer objects search their surrounding environment with more precision. Indeed, this behavior, known here as the adjustment, is similar to arranging the learning rate in a neural network. The spring's stiffness coefficient becomes smaller with the passing of time as a result of the spring's force. Another reason why the spring stiffness coefficient decreases with time is that the objects tend to concentrate around better locations and need to search space with smaller and more precise steps.

Each object can influence the radius of its neighborhood according to its fitness value. As indicated in Figure 3, each object can move as influenced by the spring forces imposed on it.

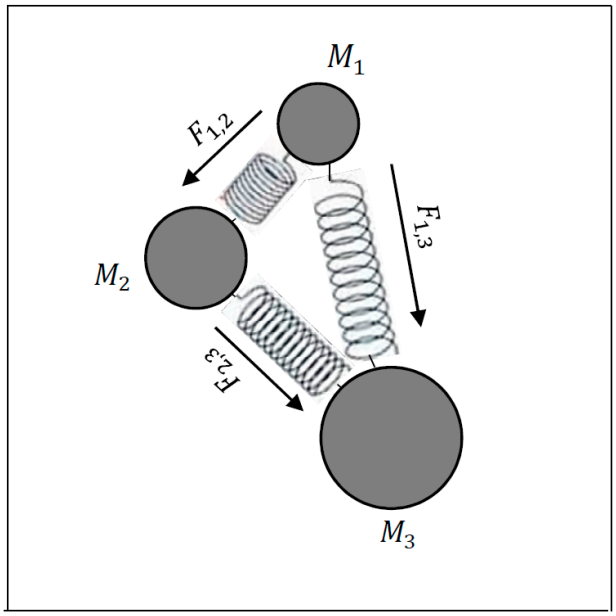

Figure 3. Forces in the system of objects and springs.

6. Exploration and Exploitation in *SSA*

The optimization method must address two issues: exploration and exploitation. In the exploration aspect, the algorithm must have enough power to search the problem search space well and not be limited to only a few specific locations. The algorithm tackles exploitation by focusing on exploring optimal locations. Before running a population algorithm, it is necessary to search the designated space comprehensively. Hence, the algorithm must focus on searching for the solution's general area during initial iterations, though as the time passes, it must locate itself more efficiently through aid from the population findings [85].

SSAs can search space by considering the suitable number of objects. The way the optimization algorithm improves its detection power is through the spring force effect by varying the spring's

stiffness coefficient and consequently controlling the spring forces among the objects. During initial iterations, the problem needs thorough searches, and as time passes, the population arrives at better results and the spring stiffness coefficient value is controlled. At the initial time, a proper value is chosen, and as time passes, that value decreases though the spring stiffness coefficient (Equation (11)) until it arrives at its minimum value.

7. Experimental Results and Discussion

This section presents the results from the evaluation of the SSA's performance on twenty-three standard benchmark test functions. A detailed description of these benchmark functions is presented below. Furthermore, the results are compared to eight existing optimization algorithms.

7.1. Benchmark Test Functions

The performance of the SSA was assessed by using 23 benchmark test functions [86]. The experimentation was done on MATLAB version R2014a (8.3.0.532) in a Microsoft Windows 7 environment using a 64 bit Core i-7 processor with 2.40 GHz and 16 GB main memory. The average and standard deviation of the best optimal solutions are displayed in Tables 1–3. For each benchmark test function, the SSA utilizes 20 independent runs, in which each run employs 1000 iterations.

7.2. Algorithms Used for Comparison

In order to prove the potency of the SSA, it is also compared to eight optimization algorithms on unimodal, multimodal, fixed-dimension multimodal and composite optimization. They were assessed by solving a set of minimization problems introduced in the Constrained Single Objective Real-Parameter Optimization Technical Report, 'CEC'2015' [86]. To validate the performance of the *SSA*, the eight optimization algorithms included: GA [87], PSO [29], GSA [73], TLBO [88], GOA [53], GWO [52], SHO [34], and EPO [58].

The parameter values of optimization algorithms are given in the next table.

Parameter definition
1: *GA*
2: *Population size N = 80*
3: *Crossover 0.9*
4: *Mutation 0.05*
5: *PSO*
6: *Swarm size S = 50*
7: *Inertia weight decreases linearly from 0.9 to 0.4*
8: *C1(individual-best acceleration factor) increases linearly from 0.5 to 2.5*
9: *C2(global-best acceleration factor) decreases linearly from 2.5 to 0.5*
10: *GSA*
11: *Objects number N = 50*
12: *Acceleration coefficient (a = 20)*
13: *Initial gravitational constant (G0 = 100)*
14: *TLBO*
15: *Swarm size S = 50*
16: *GOA*
17: *Search Agents N = 100*
18: $C_{max} = 1$
19: $C_{min} = 4 \times 10^{-5}$
20: *l = 1.5 and f = 0.5*
21: *GWO*
22: *Wolves number = 50*

23: *a variable decreases linearly from 2 to 0*
24: *SHO*
25: *Search Agents N = 80*
26: *M Constant* [0.5, 1]
27: *Control Parameter (h)* [5, 0]
28: *EPO*
29: *Search Agents N = 80*
30: *Temperature Profile* [1, 1000]
31: *A Constant* [−1.5, 1.5]
32: *Function S()* [0, 1.5]
33: *Parameter M = 2*
34: *Parameter f* [2, 3]
35: *Parameter l* [1.5, 2]

7.2.1. Evaluation of Unimodal Test Function with High Dimensions

Functions F_1 to F_7 are unimodal. The mean results of 20 independent runs of the algorithm are displayed in Table 1. These results show that the SSA has a better performance in all F_1 to F_7 functions than other algorithms.

Table 1. Results for SSA and other algorithms in Unimodal test functions.

		GA	*PSO*	*GSA*	*TLBO*	*GOA*	*GWO*	*SHO*	*EPO*	*SSA*
F_1	Ave	1.95×10^{-12}	4.98×10^{-09}	1.16×10^{-16}	3.55×10^{-02}	2.81×10^{-01}	7.86×10^{-10}	4.61×10^{-23}	5.71×10^{-28}	6.74×10^{-35}
	std	2.01×10^{-11}	1.40×10^{-08}	6.10×10^{-17}	1.06×10^{-01}	1.11×10^{-01}	8.11×10^{-09}	7.37×10^{-23}	8.31×10^{-29}	9.17×10^{-36}
F_2	Ave	6.53×10^{-18}	7.29×10^{-04}	1.70×10^{-01}	3.23×10^{-05}	3.96×10^{-01}	5.99×10^{-20}	1.20×10^{-34}	6.20×10^{-40}	7.78×10^{-45}
	std	5.10×10^{-17}	1.84×10^{-03}	9.29×10^{-01}	8.57×10^{-05}	1.41×10^{-01}	1.11×10^{-17}	1.30×10^{-34}	3.32×10^{-40}	3.48×10^{-45}
F_3	Ave	7.70×10^{-10}	1.40×10^{01}	4.16×10^{02}	4.91×10^{03}	4.31×10^{01}	9.19×10^{-05}	1.00×10^{-14}	2.05×10^{-19}	2.63×10^{-25}
	std	7.36×10^{-09}	7.13×10^{00}	1.56×10^{02}	3.89×10^{03}	8.97×10^{00}	6.16×10^{-04}	4.10×10^{-14}	9.17×10^{-20}	9.83×10^{-27}
F_4	Ave	9.17×10^{01}	6.00×10^{-01}	1.12×10^{00}	1.87×10^{01}	8.80×10^{-01}	8.73×10^{-01}	2.02×10^{-14}	4.32×10^{-18}	4.65×10^{-26}
	std	5.67×10^{01}	1.72×10^{-01}	9.89×10^{-01}	8.21×10^{00}	2.50×10^{-01}	1.19×10^{-01}	2.43×10^{-14}	3.98×10^{-19}	4.68×10^{-29}
F_5	Ave	5.57×10^{02}	4.93×10^{01}	3.85×10^{01}	7.37×10^{02}	1.18×10^{02}	8.91×10^{02}	2.79×10^{01}	5.07×10^{00}	5.41×10^{-01}
	std	4.16×10^{01}	3.89×10^{01}	3.47×10^{01}	1.98×10^{03}	1.43×10^{02}	2.97×10^{02}	1.84×10^{00}	4.90×10^{-01}	5.05×10^{-02}
F_6	Ave	3.15×10^{-01}	9.23×10^{-09}	1.08×10^{-16}	4.88×10^{00}	3.15×10^{-01}	8.18×10^{-17}	6.58×10^{-01}	7.01×10^{-19}	8.03×10^{-24}
	std	9.98×10^{-02}	1.78×10^{-08}	4.00×10^{-17}	9.75×10^{-01}	9.98×10^{-02}	1.70×10^{-18}	3.38×10^{-01}	4.39×10^{-20}	5.22×10^{-26}
F_7	Ave	6.79×10^{-04}	6.92×10^{-02}	7.68×10^{-01}	3.88×10^{-02}	2.02×10^{-02}	5.37×10^{-01}	7.80×10^{-04}	2.71×10^{-05}	3.33×10^{-08}
	std	3.29×10^{-03}	2.87×10^{-02}	2.77×10^{00}	5.79×10^{-02}	7.43×10^{-03}	1.89×10^{-01}	3.85×10^{-04}	9.26×10^{-06}	1.18×10^{-06}

7.2.2. Evaluation of Multimodal Test Functions with High Dimensions

In multimodal functions F_8 to F_{13}, by increasing the function dimensions, the number of local responses increased exponentially. Therefore, arriving at the minimum response of these functions is hard to achieve. In these types of functions, arriving at a response close to the ideal response denotes the algorithm's high capability in avoiding wrong local responses. The results obtained from assessing F_8 to F_{13} after 20 runs of the SSA as well as the other algorithms compared are presented in Table 2. The SSA demonstrated the best performance of all the functions analyzed.

7.2.3. Evaluation of Multimodal Test Functions with Low Dimensions

Functions F_{14} to F_{23} have both low dimensions in addition to low local responses. Results obtained from 20 runs of each algorithm, are shown in Table 3. These results indicate that the SSA has a suitable performance in all functions F_{14} to F_{23}.

7.2.4. Evaluation of CEC 2015 Test Functions

This section is devoted to real approaches and techniques for solving single objective optimization problems. All of these test functions are minimization problems. Table 4 shows the performance of the SSA as well as the other algorithms on the CEC 2015 test. Table 4 exhibits how the SSA is the most efficient optimizer of all of the benchmark test functions studied.

Table 2. Results for SSA and other algorithms in Multimodal test functions.

		GA	*PSO*	*GSA*	*TLBO*	*GOA*	*GWO*	*SHO*	*EPO*	*SSA*
F_8	Ave	-5.11×10^{02}	-5.01×10^{02}	-2.75×10^{02}	-3.81×10^{02}	-6.92×10^{02}	-4.69×10^{01}	-6.14×10^{02}	-8.76×10^{02}	-1.2×10^{04}
	std	4.37×10^{01}	4.28×10^{01}	5.72×10^{01}	2.83×10^{01}	9.19×10^{01}	3.94×10^{01}	9.32×10^{01}	5.92×10^{01}	9.14×10^{-12}
F_9	Ave	1.23×10^{-01}	1.20×10^{-01}	3.35×10^{01}	2.23×10^{01}	1.01×10^{02}	4.85×10^{-02}	4.34×10^{-01}	6.90×10^{-01}	8.76×10^{-04}
	std	4.11×10^{01}	4.01×10^{01}	1.19×10^{01}	3.25×10^{01}	1.89×10^{01}	3.91×10^{01}	1.66×10^{00}	4.81×10^{-01}	4.85×10^{-02}
F_{10}	Ave	5.31×10^{-11}	5.20×10^{-11}	8.25×10^{-09}	1.55×10^{01}	1.15×10^{00}	2.83×10^{-08}	1.63×10^{-14}	8.03×10^{-16}	8.04×10^{-20}
	std	1.11×10^{-10}	1.08×10^{-10}	1.90×10^{-09}	8.11×10^{00}	7.87×10^{-01}	4.34×10^{-07}	3.14×10^{-15}	2.74×10^{-14}	3.34×10^{-18}
F_{11}	Ave	3.31×10^{-06}	3.24×10^{-06}	8.19×10^{00}	3.01×10^{-01}	5.74×10^{-01}	2.49×10^{-05}	2.29×10^{-03}	4.20×10^{-05}	4.23×10^{-10}
	std	4.23×10^{-05}	4.11×10^{-05}	3.70×10^{00}	2.89×10^{-01}	1.12×10^{-01}	1.34×10^{-04}	5.24×10^{-03}	4.73×10^{-04}	5.11×10^{-07}
F_{12}	Ave	9.16×10^{-08}	8.93×10^{-08}	2.65×10^{-01}	5.21×10^{01}	1.27×10^{00}	1.34×10^{-05}	3.93×10^{-02}	5.09×10^{-03}	6.33×10^{-05}
	std	4.88×10^{-07}	4.77×10^{-07}	3.14×10^{-01}	2.47×10^{02}	1.02×10^{00}	6.23×10^{-04}	2.42×10^{-02}	3.75×10^{-03}	4.71×10^{-04}
F_{13}	Ave	6.39×10^{-02}	6.26×10^{-02}	5.73×10^{-32}	2.81×10^{02}	6.60×10^{-02}	9.94×10^{-08}	4.75×10^{-01}	1.25×10^{-08}	0.00×10^{00}
	std	4.49×10^{-02}	4.39×10^{-02}	8.95×10^{-32}	8.63×10^{02}	4.33×10^{-02}	2.61×10^{-07}	2.38×10^{-01}	2.61×10^{-07}	0.00×10^{00}

Table 3. Results for SSA and other algorithms in Multimodal test functions with low dimension.

		GA	*PSO*	*GSA*	*TLBO*	*GOA*	*GWO*	*SHO*	*EPO*	*SSA*
F_{14}	Ave	4.39×10^{00}	2.77×10^{00}	3.61×10^{00}	6.79×10^{00}	9.98×10^{01}	1.26×10^{00}	3.71×10^{00}	1.08×10^{00}	9.98×10^{-01}
	std	4.41×10^{-02}	2.32×10^{00}	2.96×10^{00}	1.12×10^{00}	9.14×10^{-1}	6.86×10^{-01}	3.86×10^{00}	4.11×10^{-02}	7.64×10^{-12}
F_{15}	Ave	7.36×10^{-02}	9.09×10^{-03}	6.84×10^{-02}	5.15×10^{-02}	7.15×10^{-02}	1.01×10^{-02}	3.66×10^{-02}	8.21×10^{-03}	3.3×10^{-04}
	std	2.39×10^{-03}	2.38×10^{-03}	7.37×10^{-02}	3.45×10^{-03}	1.26×10^{-01}	3.75×10^{-03}	7.60×10^{-02}	4.09×10^{-03}	1.25×10^{-05}
F_{16}	Ave	-1.02×10^{00}	-1.02×10^{00}	-1.02×10^{00}	-1.01×10^{00}	-1.02×10^{00}	-1.02×10^{00}	-1.02×10^{00}	-1.02×10^{00}	-1.03×10^{00}
	std	4.19×10^{-07}	0.00×10^{00}	0.00×10^{00}	3.64×10^{-08}	4.74×10^{-08}	3.23×10^{-05}	7.02×10^{-09}	9.80×10^{-07}	5.12×10^{-10}
F_{17}	Ave	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}	3.98×10^{-01}
	std	3.71×10^{-17}	9.03×10^{-16}	1.13×10^{-16}	9.45×10^{-15}	1.15×10^{-07}	7.61×10^{-04}	7.00×10^{-07}	5.39×10^{-05}	4.56×10^{-21}
F_{18}	Ave	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}	3.00×10^{00}
	std	6.33×10^{-07}	6.59×10^{-05}	3.24×10^{-02}	1.94×10^{-10}	1.48×10^{01}	2.25×10^{-05}	7.16×10^{-06}	1.15×10^{-08}	1.15×10^{-18}
F_{19}	Ave	-3.81×10^{00}	-3.80×10^{00}	-3.86×10^{00}	-3.73×10^{00}	-3.77×10^{00}	-3.75×10^{00}	-3.84×10^{00}	-3.86×10^{00}	-3.86×10^{00}
	std	4.37×10^{-10}	3.37×10^{-15}	4.15×10^{-01}	9.69×10^{-04}	3.53×10^{-07}	2.55×10^{-03}	1.57×10^{-03}	6.50×10^{-07}	5.61×10^{-10}
F_{20}	Ave	-2.39×10^{00}	-3.32×10^{00}	-1.47×10^{00}	-2.17×10^{00}	-3.23×10^{00}	-2.84×10^{00}	-3.27×10^{00}	-2.81×10^{00}	-3.31×10^{00}
	std	4.37×10^{-01}	2.66×10^{-01}	5.32×10^{-01}	1.64×10^{-01}	5.37×10^{-02}	3.71×10^{-01}	7.27×10^{-02}	7.11×10^{-01}	4.29×10^{-05}
F_{21}	Ave	-5.19×10^{00}	-7.54×10^{00}	-4570	-7.33×10^{00}	-7.38×10^{00}	-2.28×10^{00}	-9.65×10^{00}	-8.07×10^{00}	-10.15×10^{00}
	std	2.34×10^{00}	2.77×10^{00}	1.30×10^{00}	1.29×10^{00}	2.91×10^{00}	1.80×10^{00}	1.54×10^{00}	2.29×10^{00}	1.25×10^{-02}
F_{22}	Ave	-2.97×10^{00}	-8.55×10^{00}	-6.58×10^{00}	-1.00×10^{00}	-8.50×10^{00}	-3.99×10^{00}	-1.04×10^{00}	-10.01×10^{00}	-10.40×10^{00}
	std	1.37×10^{-02}	3.08×10^{00}	2.64×10^{00}	2.89×10^{-04}	3.02×10^{00}	1.99×10^{00}	2.73×10^{-04}	3.97×10^{-02}	3.65×10^{-07}
F_{23}	Ave	-3.10×10^{00}	-9.19×10^{00}	-9.37×10^{00}	-2.46×10^{00}	-8.41×10^{00}	-4.49×10^{00}	-1.05×10^{01}	-3.41×10^{00}	-10.53×10^{00}
	std	2.37×10^{00}	2.52×10^{00}	2.75×10^{00}	1.19×10^{00}	3.13×10^{00}	1.96×10^{00}	1.81×10^{-04}	1.11×10^{-02}	5.26×10^{-06}

Table 4. Results for SSA and other algorithms in CEC 2015.

		GA	*PSO*	*GSA*	*TLBO*	*GOA*	*GWO*	*SHO*	*EPO*	*SSA*
Cec-1	Ave	8.89×10^{06}	3.20×10^{07}	7.65×10^{06}	1.47×10^{06}	4.37×10^{05}	2.02×10^{06}	2.28×10^{06}	6.06×10^{05}	1.50×10^{05}
	std	6.95×10^{06}	8.37×10^{06}	3.07×10^{06}	2.63×10^{06}	4.73×10^{05}	2.08×10^{06}	2.18×10^{06}	5.02×10^{05}	1.21×10^{05}
Cec-2	Ave	2.97×10^{05}	6.58×10^{03}	7.33×10^{08}	1.97×10^{04}	9.41×10^{03}	5.65×10^{06}	3.13×10^{05}	1.43×10^{04}	4.40×10^{03}
	std	2.85×10^{03}	1.09×10^{03}	2.33×10^{08}	1.46×10^{04}	1.08×10^{04}	6.03×10^{06}	4.19×10^{05}	1.03×10^{04}	1.34×10^{08}
Cec-3	Ave	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}	3.20×10^{02}
	std	2.78×10^{-02}	1.11×10^{-05}	7.53×10^{-02}	3.19×10^{-02}	9.14×10^{-02}	8.61×10^{-02}	7.08×10^{-02}	3.76×10^{-02}	1.16×10^{-06}
Cec-4	Ave	6.99×10^{02}	4.39×10^{02}	4.42×10^{02}	4.18×10^{02}	4.26×10^{02}	4.09×10^{02}	4.16×10^{02}	4.11×10^{02}	4.04×10^{02}
	std	6.43×10^{00}	7.25×10^{00}	7.72×10^{00}	1.03×10^{01}	1.17×10^{01}	3.96×10^{00}	1.03×10^{01}	1.71×10^{01}	5.61×10^{00}
Cec-5	Ave	1.26×10^{03}	1.75×10^{03}	1.76×10^{03}	1.09×10^{03}	1.33×10^{03}	8.65×10^{02}	9.20×10^{02}	9.13×10^{02}	9.81×10^{02}
	std	1.86×10^{02}	2.79×10^{02}	2.30×10^{02}	2.81×10^{02}	3.45×10^{02}	2.16×10^{02}	1.78×10^{02}	1.85×10^{02}	1.06×10^{02}
Cec-6	Ave	2.91×10^{05}	3.91×10^{06}	2.30×10^{04}	3.82×10^{03}	7.35×10^{03}	1.86×10^{03}	2.26×10^{04}	1.29×10^{04}	1.05×10^{03}
	std	1.67×10^{05}	2.70×10^{06}	2.41×10^{04}	2.44×10^{03}	3.82×10^{03}	1.93×10^{03}	2.45×10^{04}	1.15×10^{04}	1.05×10^{03}
Cec-7	Ave	7.08×10^{02}	7.08×10^{02}	7.06×10^{02}	7.02×10^{02}	7.02×10^{02}	7.02×10^{02}	7.02×10^{02}	7.02×10^{02}	7.02×10^{02}
	std	2.97×10^{00}	1.32×10^{00}	9.07×10^{-01}	9.40×10^{-01}	1.10×10^{00}	7.75×10^{-01}	7.07×10^{-01}	6.76×10^{-01}	5.50×10^{-01}
Cec-8	Ave	5.79×10^{04}	6.07×10^{05}	6.73×10^{03}	2.58×10^{03}	9.93×10^{03}	3.43×10^{03}	3.49×10^{03}	1.86×10^{03}	1.47×10^{03}
	std	2.76×10^{04}	4.81×10^{05}	3.36×10^{03}	1.61×10^{03}	8.74×10^{03}	2.77×10^{03}	2.04×10^{03}	1.98×10^{03}	1.34×10^{03}
Cec-9	Ave	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}	1.00×10^{03}
	std	3.97×10^{00}	5.33×10^{00}	9.79×10^{-01}	5.29×10^{-02}	2.20×10^{-01}	7.23×10^{-02}	1.28×10^{-01}	1.43×10^{-01}	1.51×10^{-03}
Cec-10	Ave	4.13×10^{04}	3.42×10^{05}	9.91×10^{03}	2.62×10^{03}	8.39×10^{03}	3.27×10^{03}	4.00×10^{03}	2.00×10^{03}	1.23×10^{03}
	std	2.39×10^{04}	1.74×10^{05}	8.83×10^{03}	1.78×10^{03}	1.12×10^{04}	1.84×10^{03}	2.82×10^{03}	2.73×10^{03}	1.51×10^{03}

Table 4. *Cont.*

		GA	*PSO*	*GSA*	*TLBO*	*GOA*	*GWO*	*SHO*	*EPO*	*SSA*
Cec-11	Ave	1.36×10^{03}	1.41×10^{03}	1.35×10^{03}	1.39×10^{03}	1.37×10^{03}	1.35×10^{03}	1.40×10^{03}	1.38×10^{03}	1.35×10^{03}
	std	5.39×10^{01}	7.73×10^{01}	1.11×10^{02}	5.42×10^{01}	8.97×10^{01}	1.12×10^{02}	5.81×10^{01}	2.42×10^{01}	1.01×10^{01}
Cec-12	Ave	1.31×10^{03}	1.31×10^{03}	1.31×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}
	std	1.65×10^{00}	2.05×10^{00}	1.54×10^{00}	8.07×10^{-01}	9.14×10^{-01}	6.94×10^{-01}	6.69×10^{-01}	7.89×10^{-01}	1.50×10^{-01}
Cec-13	Ave	1.35×10^{03}	1.35×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}	1.30×10^{03}
	std	3.97×10^{01}	4.70×10^{01}	3.78×10^{-03}	2.43×10^{-04}	1.04×10^{-03}	5.44×10^{-03}	1.92×10^{-04}	2.76×10^{-04}	6.43×10^{-05}
Cec-14	Ave	8.96×10^{03}	9.30×10^{03}	7.51×10^{03}	7.34×10^{03}	7.60×10^{03}	7.10×10^{03}	7.29×10^{03}	4.25×10^{03}	3.22×10^{03}
	std	6.32×10^{03}	4.04×10^{02}	1.52×10^{03}	2.47×10^{03}	1.29×10^{03}	3.12×10^{03}	2.45×10^{03}	1.73×10^{03}	2.12×10^{02}
Cec-15	Ave	1.63×10^{03}	1.64×10^{03}	1.62×10^{03}	1.60×10^{03}	1.61×10^{03}	1.60×10^{03}	1.61×10^{03}	1.60×10^{03}	1.60×10^{03}
	std	3.67×10^{01}	1.12×10^{01}	3.64×10^{00}	1.80×10^{-02}	1.13×10^{01}	2.66×10^{00}	4.94×10^{00}	3.76×10^{00}	5.69×10^{-01}

8. SSA for Engineering Design Problems

In this section, the SSA is applied to five constrained engineering design problems.

8.1. Pressure Vessel Design

The mathematical model of this problem was adapted from a paper by Kannan and Kramer [89]. Tables 5 and 6 show the performance of the SSA along with the other algorithms. The SSA provides an optimal solution at (0.778099, 0.383241, 40.315121, 200.00000), with a corresponding fitness value of 5880.0700.

Table 5. Comparison results for pressure vessel design problem.

Algorithms	Optimum Variables				Optimum Cost
	T_s	T_h	R	L	
SSA	0.778099	0.383241	40.315121	200.00000	5880.0700
EPO	0.778210	0.384889	40.315040	200.00000	5885.5773
SHO	0.779035	0.384660	40.327793	199.65029	5889.3689
GOA	0.778961	0.384683	40.320913	200.00000	5891.3879
GWO	0.845719	0.418564	43.816270	156.38164	6011.5148
TLBO	0.817577	0.417932	41.74939	183.57270	6137.3724
GSA	1.085800	0.949614	49.345231	169.48741	11,550.2976
PSO	0.752362	0.399540	40.452514	198.00268	5890.3279
GA	1.099523	0.906579	44.456397	179.65887	6550.0230

Table 6. Statistical results for pressure vessel design problem.

Algorithms	Best	Mean	Worst	Std. Dev.	Median
SSA	5880.0700	5891.3099	024.341	5883.5153	
EPO	5885.5773	5887.4441	5892.3207	002.893	5886.2282
SHO	5889.3689	5891.5247	5894.6238	013.910	5890.6497
GOA	5891.3879	6531.5032	7394.5879	534.119	6416.1138
GWO	6011.5148	6477.3050	7250.9170	327.007	6397.4805
TLBO	6137.3724	6326.7606	6512.3541	126.609	6318.3179
GSA	11,550.2976	23,342.2909	33,226.2526	5790.625	24,010.0415
PSO	5890.3279	6264.0053	7005.7500	496.128	6112.6899
GA	6550.0230	6643.9870	8005.4397	657.523	7586.0085

8.2. Speed Reducer Design Problem

This problem is modeled mathematically in [90,91]. The results of the optimization problem are presented in Tables 7 and 8. The optimal solution was provided by the SSA at (3.50123, 0.7, 17, 7.3, 7.8, 3.33421, 5.26536) with a corresponding fitness value equal to 2994.2472.

Table 7. Comparison results for speed reducer design problem.

Algorithms	Optimum Variables							Optimum Cost
	b	m	p	l_1	l_2	d_1	d_2	
SSA	3.50123	0.7	17	7.3	7.8	3.33421	5.26536	2994.2472
EPO	3.50159	0.7	17	7.3	7.8	3.35127	5.28874	2998.5507
SHO	3.506690	0.7	17	7.380933	7.815726	3.357847	5.286768	3001.288
GOA	3.500019	0.7	17	8.3	7.8	3.352412	5.286715	3005.763
GWO	3.508502	0.7	17	7.392843	7.816034	3.358073	5.286777	3002.928
TLBO	3.508755	0.7	17	7.3	7.8	3.461020	5.289213	3030.563
GSA	3.600000	0.7	17	8.3	7.8	3.369658	5.289224	3051.120
PSO	3.510253	0.7	17	8.35	7.8	3.362201	5.287723	3067.561
GA	3.520124	0.7	17	8.37	7.8	3.366970	5.288719	3029.002

Table 8. Statistical results for speed reducer design problem.

Algorithms	Best	Mean	Worst	Std. Dev.	Median
SSA	2994.2472	2997.482	2999.092	1.78091	2996.318
EPO	2998.5507	2999.640	3003.889	1.93193	2999.187
SHO	3001.288	3005.845	3008.752	5.83794	3004.519
GOA	3005.763	3105.252	3211.174	79.6381	3105.252
GWO	3002.928	3028.841	3060.958	13.0186	3027.031
TLBO	3030.563	3065.917	3104.779	18.0742	3065.609
GSA	3051.120	3170.334	3363.873	92.5726	3156.752
PSO	3067.561	3186.523	3313.199	17.1186	3198.187
GA	3029.002	3295.329	3619.465	57.0235	3288.657

8.3. Welded Beam Design

The mathematical model of a welded beam design was adapted from [31]. The results to this optimization problem are presented in Tables 9 and 10. The SSA provides an optimal solution at (0.205411, 3.472341, 9.035215, 0.201153) with a corresponding fitness value equal: 1.723589.

Table 9. Comparison results for welded beam design problem.

Algorithms	Optimum Variables				Optimum Cost
	h	l	t	b	
SSA	0.205411	3.472341	9.035215	0.201153	1.723589
EPO	0.205563	3.474846	9.035799	0.205811	1.725661
SHO	0.205678	3.475403	9.036964	0.206229	1.726995
GOA	0.197411	3.315061	10.00000	0.201395	1.820395
GWO	0.205611	3.472103	9.040931	0.205709	1.725472
TLBO	0.204695	3.536291	9.004290	0.210025	1.759173
GSA	0.147098	5.490744	10.00000	0.217725	2.172858
PSO	0.164171	4.032541	10.00000	0.223647	1.873971
GA	0.206487	3.635872	10.00000	0.203249	1.836250

8.4. Tension/Compression Spring Design Problem

The mathematical model of this problem was adapted from [31]. The results to this optimization problem are displayed in Tables 11 and 12. The SSA provides the optimal solution at (0.051087, 0.342908, 12.0898), with a corresponding fitness value of 0.012656987.

Table 10. Statistical results for welded beam design problem.

Algorithms	Best	Mean	Worst	Std. Dev.	Median
SSA	1.723589	1.725124	1.727211	0.004325	1.724399
EPO	1.725661	1.725828	1.726064	0.000287	1.725787
SHO	1.726995	1.727128	1.727564	0.001157	1.727087
GOA	1.820395	2.230310	3.048231	0.324525	2.244663
GWO	1.725472	1.729680	1.741651	0.004866	1.727420
TLBO	1.759173	1.817657	1.873408	0.027543	1.820128
GSA	2.172858	2.544239	3.003657	0.255859	2.495114
PSO	1.873971	2.119240	2.320125	0.034820	2.097048
GA	1.836250	1.363527	2.035247	0.139485	1.9357485

Table 11. Comparison results for tension/compression spring design problem.

Algorithms	Optimum Variables			Optimum Cost
	d	*D*	*p*	
SSA	0.051087	0.342908	12.0898	0.012656987
EPO	0.051144	0.343751	12.0955	0.012674000
SHO	0.050178	0.341541	12.07349	0.012678321
GOA	0.05000	0.310414	15.0000	0.013192580
GWO	0.05000	0.315956	14.22623	0.012816930
TLBO	0.050780	0.334779	12.72269	0.012709667
GSA	0.05000	0.317312	14.22867	0.012873881
PSO	0.05010	0.310111	14.0000	0.013036251
GA	0.05025	0.316351	15.23960	0.012776352

Table 12. Statistical results for tension/compression spring design problem.

Algorithms	Best	Mean	Worst	Std. Dev.	Median
SSA	0.012656987	0.012678903	0.012667902	0.001021	0.012676002
EPO	0.012674000	0.012684106	0.012715185	0.000027	0.012687293
SHO	0.012678321	0.012697116	0.012720757	0.000041	0.012699686
GOA	0.013192580	0.014817181	0.017862507	0.002272	0.013192580
GWO	0.012816930	0.014464372	0.017839737	0.001622	0.014021237
TLBO	0.012709667	0.012839637	0.012998448	0.000078	0.012844664
GSA	0.012873881	0.013438871	0.014211731	0.000287	0.013367888
PSO	0.013036251	0.014036254	0.016251423	0.002073	0.013002365
GA	0.012776352	0.013069872	0.015214230	0.000375	0.012952142

8.5. Rolling Element Bearing Design Problem

The mathematical model of this problem is adapted from [92]. The results of this optimization problem are included in Tables 13 and 14 and prove that the SSA provides an optimal solution at (125, 21.41890, 10.94113, 0.515, 0.515, 0.4, 0.7, 0.3, 0.02, 0.6) with a corresponding fitness value equal to 85,067.983.

Table 13. Comparison results for rolling element bearing design problem.

Algorithms	Optimum Variables										Opt. Cost
	D_m	D_b	*Z*	f_i	f_o	*KDmin*	*KDmax*	ε	*e*	ζ	
SSA	125	21.41890	10.94113	0.515	0.515	0.4	0.7	0.3	0.02	0.6	85,067.983
EPO	125	21.40732	10.93268	0.515	0.515	0.4	0.7	0.3	0.02	0.6	85,054.532
SHO	125.6199	21.35129	10.98781	0.515	0.515	0.5	0.68807	0.300151	0.03254	0.62701	84,807.111
GOA	125	20.75388	11.17342	0.515	0.515000	0.5	0.61503	0.300000	0.05161	0.60000	81,691.202
GWO	125.6002	21.32250	10.97338	0.515	0.515000	0.5	0.68782	0.301348	0.03617	0.61061	84,491.266
TLBO	125	21.14834	10.96928	0.515	0.515	0.5	0.7	0.3	0.02778	0.62912	83,431.117
GSA	125	20.85417	11.14989	0.515	0.517746	0.5	0.61827	0.304068	0.02000	0.624638	82,276.941
PSO	125	20.77562	11.01247	0.515	0.515000	0.5	0.61397	0.300000	0.05004	0.610001	82,773.982
GA	125	20.87123	11.16697	0.515	0.516000	0.5	0.61951	0.301128	0.05024	0.614531	81,569.527

Table 14. Statistical results for Rolling element bearing design problem.

Algorithms	Best	Mean	Worst	Std. Dev.	Median
SSA	85,067.983	85,042.352	86,551.599	1877.09	85,056.095
EPO	85,054.532	85,024.858	85,853.876	0186.68	85,040.241
SHO	84,807.111	84,791.613	84,517.923	0137.186	84,960.147
GOA	81,691.202	50,435.017	32,761.546	13,962.150	42,287.581
GWO	84,491.266	84,353.685	84,100.834	0392.431	84,398.601
TLBO	83,431.117	81,005.232	77,992.482	1710.777	81,035.109
GSA	82,276.941	78,002.107	71,043.110	3119.904	78,398.853
PSO	82,773.982	81,198.753	80,687.239	1679.367	8439.728
GA	81,569.527	80,397.998	79,412.779	1756.902	8347.009

9. Conclusions

There are many optimization problems in the different scientific domains that must be solved using the algorithms with each case's necessary characteristics. In this project, a new optimization algorithm called spring search algorithm (SSA) was developed; Hooke's law describes the starting point or basic concept. The search agents of the proposed method are weights that are connected by several springs. The system starts from a transitory situation or state and stabilizes at the equilibrium point, according to the law of the spring.

To evaluate the algorithm and purchase it, almost 40 standard objective functions, including unimodal and multimodal functions, in addition to CEC2015, were used to assess the performance of the proposed algorithm in solving optimization problems of a different nature. To review and analyze the algorithm's results, these were compared with eight widely known optimization algorithms: GA, PSO, GSA, TLBO, GWO, GOA, SHO, and EPO.

The results in the functions show the superior exploration and exploitation capabilities of SSA compared to other optimization algorithms, for both unimodal and multimodal functions. The same occurs with the simulations using the SSA algorithm and the eight algorithms selected for comparison in the case of CEC2015, which shows the SSA's high aptitude to solve this type of problem to optimize the function. In addition to the work carried out on the almost 40 functions, the SSA algorithm was evaluated in five engineering design optimization problems to evaluate the performance in solving optimization problems in real situations, showing that it is much more competitive than other algorithms.

For future work, it is suggested to develop a binary version of the SSA algorithm and apply this algorithm to multi-objective problems. Likewise, extending the concept of Hooke's law to more complex models with more adjustment parameters, which, even though it will reduce simplicity, could generate additional advantages.

Author Contributions: Conceptualization, M.D., Z.M., R.M.-M., R.A.R.-M., A.D. and J.M.G.; methodology, M.D. and Z.M.; software, M.D.; validation, J.M.G., G.D., R.A.R.-M., R.M.-M., A.D. and O.P.M.; formal analysis, A.D., O.P.M.; investigation, M.D. and O.P.M.; resources, J.M.G.; data curation, G.D.; writing—original draft preparation, M.D. and Z.M.; writing—review and editing, O.P.M., R.A.R.-M., R.M.-M., L.P.-A., G.D., and J.M.G.; visualization, M.D.; supervision, M.D. and Z.M.; project administration, M.D. and Z.M.; funding acquisition, R.A.R.-M. and R.M.-M. All authors have read and agreed to the published version of the manuscript.

Funding: The current project was funded by Tecnologico de Monterrey and FEMSA Foundation (grant CAMPUSCITY project).

Conflicts of Interest: The authors declare no conflict of interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Abbreviations

Acronym	Definition
ABC	Artificial Bee Colony
ACROA	Artificial Chemical Reaction Optimization Algorithm
AFSA	Artificial Fish-Swarm Algorithm
BA	Bat-inspired Algorithm
BBO	Biogeography-Based Optimizer
BH	Black Hole
BOSA	Binary Orientation Search Algorithm
BBBC	Big-Bang Big-Crunch
CFO	Central Force Optimization
CS	Cuckoo Search
CSO	Curved Space Optimization
CSS	Charged System Search
DGO	Darts Game Optimizer
DPO	Dolphin Partner Optimization
DGO	Dice Game Optimizer
DE	Differential Evolution
DTO	Donkey Theorem Optimization
EP	Evolutionary Programming
ES	Evolution Strategy
EPO	Emperor Penguin Optimizer
FA	Firefly Algorithm
FOA	Following Optimization Algorithm
FGBO	Football Game Based Optimization
GP	Genetic Programming
GO	Group Optimization
GOA	Grasshopper Optimization Algorithm
GSA	Gravitational Search Algorithm
GbSA	Galaxy-based Search Algorithm
GWO	Grey Wolf Optimizer
HOGO	Hide Objects Game Optimization
HS	Hunting Search
MFO	Moth-flame Optimization Algorithm
MS	Monkey Search
OSA	Orientation Search Algorithm
PSO	Particle Swarm Optimization
RSO	Rat Swarm Optimizer
RO	Ray Optimization
SHO	Spotted Hyena Optimizer
SGO	Shell Game Optimization
SWOA	Small World Optimization Algorithm
WOA	Whale Optimization Algorithm

References

1. Cortés-Toro, E.M.; Crawford, B.; Gómez-Pulido, J.A.; Soto, R.; Lanza-Gutiérrez, J.M. A new metaheuristic inspired by the vapour-liquid equilibrium for continuous optimization. *Appl. Sci.* **2018**, *8*, 2080. [CrossRef]
2. Pelusi, D.; Mascella, R.; Tallini, L. A fuzzy gravitational search algorithm to design optimal IIR filters. *Energies* **2018**, *11*, 736. [CrossRef]
3. Díaz, P.; Pérez-Cisneros, M.; Cuevas, E.; Avalos, O.; Gálvez, J.; Hinojosa, S.; Zaldivar, D. An improved crow search algorithm applied to energy problems. *Energies* **2018**, *11*, 571. [CrossRef]
4. Chiu, C.-Y.; Shih, P.-C.; Li, X. A dynamic adjusting novel global harmony search for continuous optimization problems. *Symmetry* **2018**, *10*, 337. [CrossRef]

5. Sengupta, S.; Basak, S.; Peters, R.A. Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives. *Mach. Learn. Knowl. Extr.* **2019**, *1*, 157–191. [CrossRef]
6. Stripling, E.; Broucke, S.V.; Antonio, K.; Baesens, B.; Snoeck, M. Profit maximizing logistic model for customer churn prediction using genetic algorithms. *Swarm Evol. Comput.* **2018**, *40*, 116–130. [CrossRef]
7. Antonov, I.V.; Mazurov, E.; Borodovsky, M.; Medvedeva, Y.A. Prediction of lncRNAs and their interactions with nucleic acids: Benchmarking bioinformatics tools. *Brief. Bioinform.* **2019**, *20*, 551–564. [CrossRef]
8. Djenouri, Y.; Belhadi, A.; Belkebir, R. Bees swarm optimization guided by data mining techniques for document information retrieval. *Expert Syst. Appl.* **2018**, *94*, 126–136. [CrossRef]
9. Artrith, N.; Urban, A.; Ceder, G. Constructing first-principles phase diagrams of amorphous Li x Si using machine-learning-assisted sampling with an evolutionary algorithm. *J. Chem. Phys.* **2018**, *148*, 241711. [CrossRef]
10. Dehghani, M.; Montazeri, Z.; Malik, O. Energy commitment: A planning of energy carrier based on energy consumption. Электротехника и электромеханика **2019**, *4*, 69–72. [CrossRef]
11. Ehsanifar, A.; Dehghani, M.; Allahbakhshi, M. Calculating the leakage inductance for transformer inter-turn fault detection using finite element method. In Proceedings of the 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2–4 May 2017; pp. 1372–1377.
12. Dehghani, M.; Montazeri, Z.; Malik, O. Optimal sizing and placement of capacitor banks and distributed generation in distribution systems using spring search algorithm. *Int. J. Emerg. Electr. Power Syst.* **2020**, 21. [CrossRef]
13. Dehghani, M.; Montazeri, Z.; Malik, O.P.; Al-Haddad, K.; Guerrero, J.M.; Dhiman, G. A New Methodology Called Dice Game Optimizer for Capacitor Placement in Distribution Systems. Электротехника и электромеханика **2020**, *1*, 61–64. [CrossRef]
14. Dehbozorgi, S.; Ehsanifar, A.; Montazeri, Z.; Dehghani, M.; Seifi, A. Line loss reduction and voltage profile improvement in radial distribution networks using battery energy storage system. In Proceedings of the 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 22 December 2017; pp. 215–219.
15. Montazeri, Z.; Niknam, T. Optimal utilization of electrical energy from power plants based on final energy consumption using gravitational search algorithm. Электротехника и электромеханика **2018**, *4*, 70–73. [CrossRef]
16. Dehghani, M.; Mardaneh, M.; Montazeri, Z.; Ehsanifar, A.; Ebadi, M.; Grechko, O. Spring search algorithm for simultaneous placement of distributed generation and capacitors. Электротехника и электромеханика **2018**, *6*, 68–73. [CrossRef]
17. Dehghani, M.; Montazeri, Z.; Ehsanifar, A.; Seifi, A.; Ebadi, M.; Grechko, O. Planning of energy carriers based on final energy consumption using dynamic programming and particle swarm optimization. Электротехника и электромеханика **2018**, *5*, 62–71. [CrossRef]
18. Montazeri, Z.; Niknam, T. Energy carriers management based on energy consumption. In Proceedings of the2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 22 December 2017; pp. 539–543.
19. Mirjalili, S. Genetic Algorithm. In *Evolutionary Algorithms and Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 43–55.
20. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef]
21. Farmer, J.D.; Packard, N.H.; Perelson, A.S. The immune system, adaptation, and machine learning. *Phys. D Nonlinear Phenom.* **1986**, *22*, 187–204. [CrossRef]
22. Mirjalili, S. Ant Colony Optimisation. In *Evolutionary Algorithms and Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 33–42.
23. Mirjalili, S. Particle Swarm Optimisation. In *Evolutionary Algorithms and Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 15–31.
24. Dehghani, M.; Montazeri, Z.; Dehghani, A.; Seifi, A. Spring search algorithm: A new meta-heuristic optimization algorithm inspired by Hooke's law. In Proceedings of the 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 22 December 2017; pp. 210–214.

25. Mirjalili, S. Biogeography-Based Optimisation. In *Evolutionary Algorithms and Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 57–72.
26. Gigerenzer, G.; Gaissmaier, W. Heuristic decision making. *Annu. Rev. Psychol.* **2011**, *62*, 451–482. [CrossRef]
27. Lim, S.M.; Leong, K.Y. A Brief Survey on Intelligent Swarm-Based Algorithms for Solving Optimization Problems. In *Nature-inspired Methods for Stochastic, Robust and Dynamic Optimization*; IntechOpen: London, UK, 2018.
28. Kennedy, J.; Eberhart, R. Particle swarm optimization, proceeding of the IEEE International Conference on Neural Networks, Perth, Australia. *IEEE Serv. Cent. Piscataway* **1942**, *1948*, 1995.
29. Yang, X.-S. Firefly algorithm, stochastic test functions and design optimization. *arXiv* **2010**, arXiv:1003.1409.
30. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
31. Karaboga, D.; Basturk, B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization. In *Problems, LNCS: Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing*; Springer: Berlin/Heidelberg, Germany, 2007.
32. Yang, X.-S. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 65–74.
33. Dhiman, G.; Kumar, V. Spotted hyena optimizer: A novel bio-inspired based metaheuristic technique for engineering applications. *Adv. Eng. Softw.* **2017**, *114*, 48–70. [CrossRef]
34. Gandomi, A.H.; Yang, X.-S.; Alavi, A.H. Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Eng. Comput.* **2013**, *29*, 17–35. [CrossRef]
35. Mucherino, A.; Seref, O. Monkey search: A novel metaheuristic search for global optimization. *AIP Conf. Proc.* **2007**, *953*, 162–173.
36. Dehghani, M.; Montazeri, Z.; Dehghani, A.; Malik, O.P. GO: Group Optimization. *Gazi Univ. J. Sci.* **2020**, *33*, 381–392. [CrossRef]
37. Neshat, M.; Sepidnam, G.; Sargolzaei, M.; Toosi, A.N. Artificial fish swarm algorithm: A survey of the state-of-the-art, hybridization, combinatorial and indicative applications. *Artif. Intell. Rev.* **2014**, *42*, 965–997. [CrossRef]
38. Oftadeh, R.; Mahjoob, M.; Shariatpanahi, M. A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Comput. Math. Appl.* **2010**, *60*, 2087–2098. [CrossRef]
39. Mirjalili, S. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowl. Based Syst.* **2015**, *89*, 228–249. [CrossRef]
40. Shiqin, Y.; Jianjun, J.; Guangxing, Y. A dolphin partner optimization. In Proceedings of the Global Congress on Intelligent Systems, Xiamen, China, 19–21 May 2009; pp. 124–128.
41. Dehghani, M.; Montazeri, Z.; Malik, O.P.; Ehsanifar, A.; Dehghani, A. OSA: Orientation Search Algorithm. *Int. J. Ind. Electron. Control Optim.* **2019**, *2*, 99–112.
42. Dehghani, M.; Montazeri, Z.; Malik, O.P.; Dhiman, G.; Kumar, V. BOSA: Binary Orientation Search Algorithm. *Int. J. Innov. Technol. Explor. Eng. (IJITEE)* **2019**, *9*, 5306–5310.
43. Dehghani, M.; Montazeri, Z.; Malik, O.P. DGO: Dice Game Optimizer. *Gazi Univ. J. Sci.* **2019**, *32*, 871–882. [CrossRef]
44. Mohammad, D.; Zeinab, M.; Malik, O.P.; Givi, H.; Guerrero, J.M. Shell Game Optimization: A Novel Game-Based Algorithm. *Int. J. Intell. Eng. Syst.* **2020**, *13*, 246–255.
45. Dehghani, M.; Montazeri, Z.; Saremi, S.; Dehghani, A.; Malik, O.P.; Al-Haddad, K.; Guerrero, J.M. HOGO: Hide Objects Game Optimization. *Int. J. Intell. Eng. Syst.* **2020**, *13*, 216–225. [CrossRef]
46. Dehghani, M.; Mardaneh, M.; Malik, O.P.; NouraeiPour, S.M. DTO: Donkey Theorem Optimization. In Proceedings of the 2019 27th Iranian Conference on Electrical Engineering (ICEE), Yazd, Iran, 30 April–2 May 2019; pp. 1855–1859.
47. Dehghani, M.; Mardaneh, M.; Malik, O. FOA: 'Following'Optimization Algorithm for solving Power engineering optimization problems. *J. Oper. Autom. Power Eng.* **2020**, *8*, 57–64.
48. Dhiman, G.; Garg, M.; Nagar, A.K.; Kumar, V.; Dehghani, M. A Novel Algorithm for Global Optimization: Rat Swarm Optimizer. *J. Ambient Intell. Humaniz. Comput.* **2020**, *1*, 1–6.
49. Dehghani, M.; Montazeri, Z.; Givi, H.; Guerrero, J.M.; Dhiman, G. Darts Game Optimizer: A New Optimization Technique Based on Darts Game. *Int. J. Intell. Eng. Syst.* **2020**, *13*, 286–294.
50. Dehghani, M.; Mardaneh, M.; Guerrero, J.M.; Malik, O.P.; Kumar, V. Football Game Based Optimization: An Application to Solve Energy Commitment Problem. *Int. J. Intell. Eng. Syst.* **2020**, *13*, 514–523. [CrossRef]

51. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
52. Saremi, S.; Mirjalili, S.; Lewis, A. Grasshopper optimisation algorithm: Theory and application. *Adv. Eng. Softw.* **2017**, *105*, 30–47. [CrossRef]
53. Zhang, H.; Hui, Q. A Coupled Spring Forced Bat Searching Algorithm: Design, Analysis and Evaluation. In Proceedings of the 2020 American Control Conference (ACC), Denver, CO, USA, 1–3 July 2020; pp. 5016–5021.
54. Wang, Z.-J.; Zhan, Z.-H.; Kwong, S.; Jin, H.; Zhang, J. Adaptive Granularity Learning Distributed Particle Swarm Optimization for Large-Scale Optimization. *IEEE Trans. Cybern.* **2020**, 1–14. [CrossRef]
55. Dehghani, M.; Montazeri, Z.; Dehghani, A.; Ramirez-Mendoza, R.A.; Samet, H.; Guerrero, J.M.; Dhiman, G. MLO: Multi Leader Optimizer. *Int. J. Intell. Eng. Syst.* **2020**, *1*, 1–11.
56. Dehghani, M.; Mardaneh, M.; Guerrero, J.M.; Malik, O.P.; Ramirez-Mendoza, R.A.; Matas, J.; Vasquez, J.C.; Parra-Arroyo, L. A New "Doctor and Patient" Optimization Algorithm: An Application to Energy Commitment Problem. *Appl. Sci.* **2020**, *10*, 5791. [CrossRef]
57. Dhiman, G.; Kumar, V. Emperor Penguin Optimizer: A Bio-inspired Algorithm for Engineering Problems. *Knowl. Based Syst.* **2018**, *159*, 20–50. [CrossRef]
58. Karkalos, N.E.; Markopoulos, A.P.; Davim, J.P. Evolutionary-Based Methods. In *Computational Methods for Application in Industry 4.0*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 11–31.
59. Mirjalili, S. Introduction to Evolutionary Single-Objective Optimisation. In *Evolutionary Algorithms and Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 3–14.
60. Holland, J.H. Genetic Algorithms. *Sci. Am.* **1992**, *267*, 66–73. [CrossRef]
61. Bageley, J. The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms. Ph.D. Thesis, University of Michigan, Ann Arbor, MI, USA, 1967.
62. Bose, A.; Biswas, T.; Kuila, P. A Novel Genetic Algorithm Based Scheduling for Multi-core Systems. In *Smart Innovations in Communication and Computational Sciences*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 45–54.
63. Das, S.; Suganthan, P.N. Differential evolution: A survey of the state-of-the-art. *IEEE Trans. Evol. Comput.* **2011**, *15*, 4–31. [CrossRef]
64. Storn, R.; Price, K. *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*; ICSI: Berkeley, CA, USA, 1995.
65. Chakraborty, U.K. *Advances in Differential Evolution*; Springer: Berlin/Heidelberg, Germany, 2008; Volume 143.
66. Fogel, L.J.; Owens, A.J.; Walsh, M.J. *Artificial Intelligence through Simulated Evolution*; Wiley: New York, NY, USA, 1966.
67. Simon, D. Biogeography-based optimization. *IEEE Trans. Evol. Comput.* **2008**, *12*, 702–713. [CrossRef]
68. Deng, W.; Liu, H.; Xu, J.; Zhao, H.; Song, Y. An improved quantum-inspired differential evolution algorithm for deep belief network. *IEEE Trans. Instrum. Meas.* **2020**. [CrossRef]
69. Koza, J.R. Genetic programming as a means for programming computers by natural selection. *Stat. Comput.* **1994**, *4*, 87–112. [CrossRef]
70. Beyer, H.-G.; Schwefel, H.-P. Evolution strategies—A comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52. [CrossRef]
71. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. A heuristic algorithm and simulation approach to relative location of facilities. *Optim. Simulated Annealing* **1983**, *220*, 671–680.
72. Rashedi, E.; Nezamabadi-Pour, H.; Saryazdi, S. GSA: A gravitational search algorithm. *Inf. Sci.* **2009**, *179*, 2232–2248. [CrossRef]
73. Rashedi, E.; Rashedi, E.; Nezamabadi-pour, H. A comprehensive survey on gravitational search algorithm. *Swarm Evol. Comput.* **2018**, *41*, 141–158. [CrossRef]
74. Kaveh, A.; Talatahari, S. A novel heuristic optimization method: Charged system search. *Acta Mech.* **2010**, *213*, 267–289. [CrossRef]
75. Shah-Hosseini, H. Principal components analysis by the galaxy-based search algorithm: A novel metaheuristic for continuous optimisation. *Int. J. Comput. Sci. Eng.* **2011**, *6*, 132–140.
76. Moghaddam, F.F.; Moghaddam, R.F.; Cheriet, M. Curved space optimization: A random search based on general relativity theory. *arXiv* **2012**, arXiv:1208.2214.
77. Kaveh, A.; Khayatazad, M. A new meta-heuristic method: Ray optimization. *Comput. Struct.* **2012**, *112*, 283–294. [CrossRef]

78. Alatas, B. ACROA: Artificial chemical reaction optimization algorithm for global optimization. *Expert Syst. Appl.* **2011**, *38*, 13170–13180. [CrossRef]
79. Du, H.; Wu, X.; Zhuang, J. Small-world optimization algorithm for function optimization. In *International Conference on Natural Computation*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 264–273.
80. Formato, R.A. Central force optimization: A new nature inspired computational framework for multidimensional search and optimization. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 221–238.
81. Hatamlou, A. Black hole: A new heuristic optimization approach for data clustering. *Inf. Sci.* **2013**, *222*, 175–184. [CrossRef]
82. Erol, O.K.; Eksin, I. A new optimization method: Big bang–big crunch. *Adv. Eng. Softw.* **2006**, *37*, 106–111. [CrossRef]
83. Halliday, D.; Resnick, R.; Walker, J. *Fundamentals of Physics*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
84. Eiben, A.E.; Schippers, C.A. On evolutionary exploration and exploitation. *Fundam. Inform.* **1998**, *35*, 35–50. [CrossRef]
85. Yao, X.; Liu, Y.; Lin, G. Evolutionary programming made faster. *IEEE Trans. Evol. Comput.* **1999**, *3*, 82–102.
86. Chen, Q.; Liu, B.; Zhang, Q.; Liang, J.; Suganthan, P.; Qu, B. *Problem Definition and Evaluation Criteria for CEC 2015 Special Session and Competition on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization*; Technical Report; Computational Intelligence Laboratory, Zhengzhou University, China and Nanyang Technological University: Singapore, 2014.
87. Tang, K.-S.; Man, K.-F.; Kwong, S.; He, Q. Genetic algorithms and their applications. *IEEE Signal Process. Mag.* **1996**, *13*, 22–37. [CrossRef]
88. Sarzaeim, P.; Bozorg-Haddad, O.; Chu, X. Teaching-Learning-Based Optimization (TLBO) Algorithm. In *Advanced Optimization by Nature-Inspired Algorithms*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 51–58.
89. Kannan, B.; Kramer, S.N. An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *J. Mech. Des.* **1994**, *116*, 405–411. [CrossRef]
90. Gandomi, A.H.; Yang, X.-S. Benchmark problems in structural optimization. In *Computational Optimization, Methods and Algorithms*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 259–281.
91. Mezura-Montes, E.; Coello, C.A.C. Useful infeasible solutions in engineering optimization with evolutionary algorithms. In *Mexican International Conference on Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 652–662.
92. Rao, B.R.; Tiwari, R. Optimum design of rolling element bearings using genetic algorithms. *Mech. Mach. Theory* **2007**, *42*, 233–250.